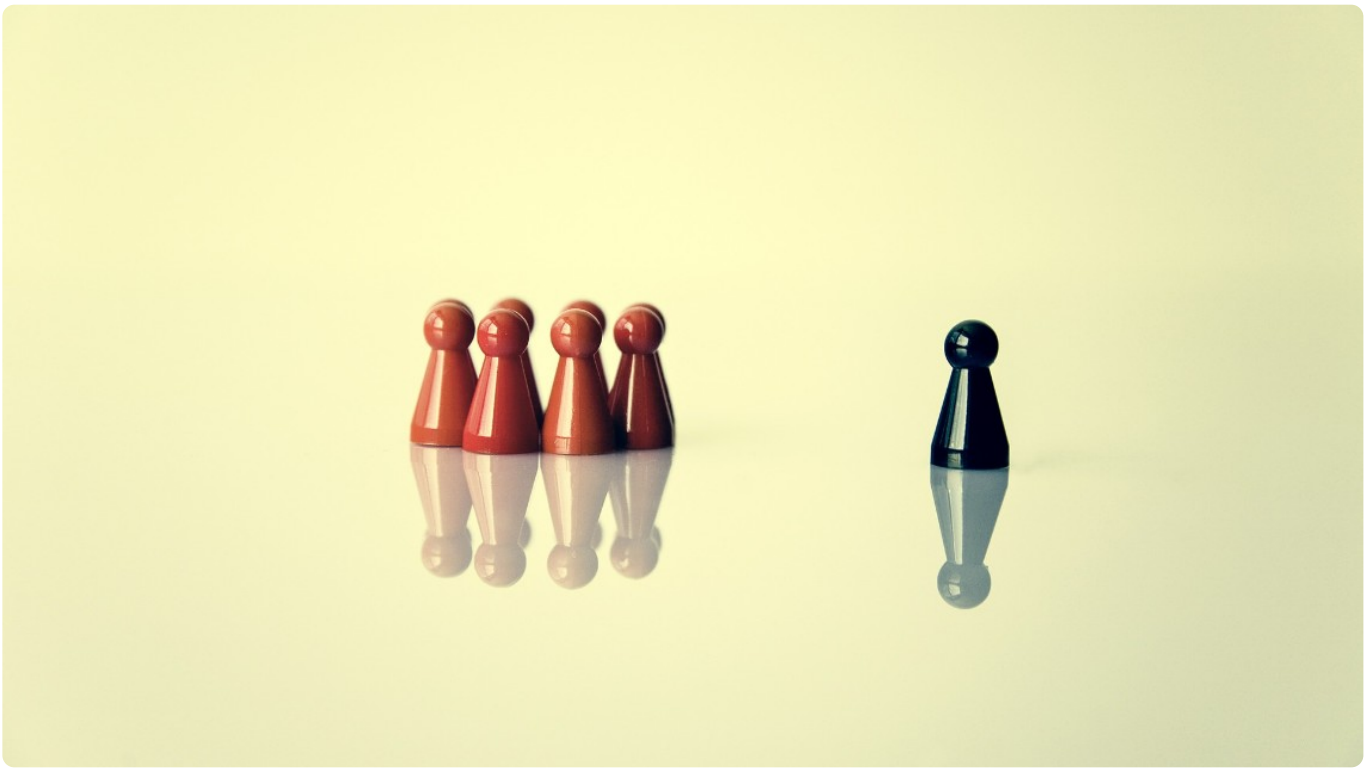
03 | 分布式互斥：有你没我，有我没你

2019-09-27 聂鹏程

分布式技术原理与算法解析 进入课程 >



讲述：聂鹏程
时长 17:16 大小 15.83M

你好！我是聂鹏程。今天，我来继续带你打卡分布式核心技术。

通过前面的两篇文章，相信你已经对此次的分布式世界之行有了一个初步了解，想必对此次旅行也是充满期待。今天，我就带你正式踏上第一站：分布式协调与同步。在这一站，我将带你学习如何让分布在不同计算机上的程序具有“团队精神”，换句话说就是如何让程序通过协作共同去达成一个业务目标。

在本站，我将带你打卡的第一个景点是分布式互斥。那，什么是分布式互斥呢？

什么是分布式互斥？

想象一下，你正在一家餐厅使用自助咖啡机泡制咖啡，突然有个人过来挪走了你的杯子，开始泡制他自己的咖啡。你耐着性子等他操作完，继续泡制自己的咖啡。结果你开始没多久，他又回来中断了你泡制咖啡的过程。相信要不了几个回合，你和他就会上演一场“有你没我，有我没你”的格斗了。

这样现实的问题也同样存在于分布式世界。就像我们使用自助咖啡机时不希望被打扰一样，对于同一共享资源，一个程序正在使用的时候也不希望被其他程序打扰。这，就要求同一时刻只能有一个程序能够访问这种资源。

在分布式系统里，这种排他性的资源访问方式，叫作**分布式互斥（Distributed Mutual Exclusion），而这种被互斥访问的共享资源就叫作临界资源（Critical Resource）。**

接下来，我带你一起看看如何才能让分布式系统里的程序互斥地访问临界资源。

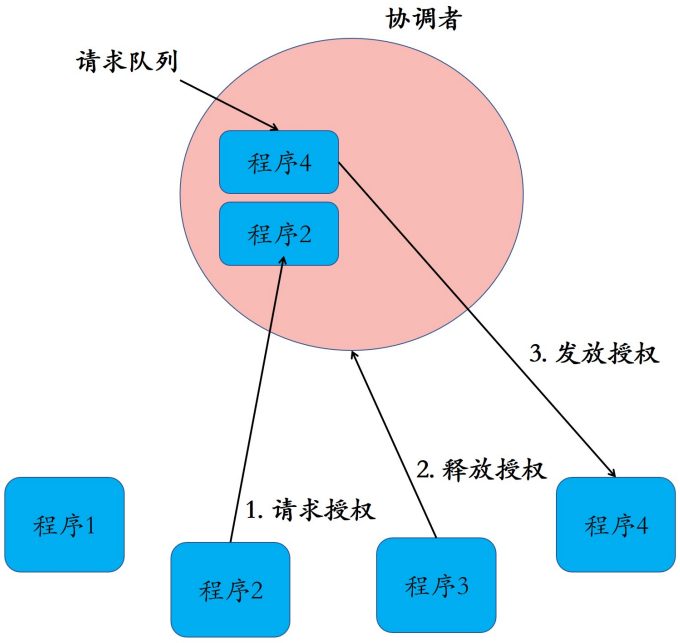
霸道总裁：集中式算法

对于前面提到的咖啡机问题，我们首先想到的就是，增加一个“协调者”来约束大家使用自助咖啡机，解决强行插入打断别人的问题。

类似地，我们引入一个协调者程序，得到一个分布式互斥算法。每个程序在需要访问临界资源时，先给协调者发送一个请求。如果当前没有程序使用这个资源，协调者直接授权请求程序访问；否则，按照先来后到的顺序为请求程序“排一个号”。如果有程序使用完资源，则通知协调者，协调者从“排号”的队列里取出排在最前面的请求，并给它发送授权消息。拿到授权消息的程序，可以直接去访问临界资源。

这个互斥算法，就是我们所说的**集中式算法**，也可以叫做中央服务器算法。之所以这么称呼，是因为协调者代表着集中程序或中央服务器。

集中式算法的示意图如下所示：



集中式算法示意图

如图所示，程序 1、2、3、4 为普通运行程序，另一个程序为协调者。当程序 2 和程序 4 需要使用临界资源时，它们会向协调者发起申请，请求协调者授权。

不巧的是，程序 3 正在使用临界资源。这时，协调者根据程序 2 和 4 的申请时间顺序，依次将它们放入等待队列。在这个案例里，程序 4 的申请时间早于程序 2，因此排在程序 2 的前面。

程序 3 使用完临界资源后，通知协调者释放授权。此时，协调者从等待队列中取出程序 4，并给它发放授权。这时，程序 4 就可以使用临界资源了。

从上述流程可以看出，**一个程序完成一次临界资源访问，需要如下几个流程和消息交互**：

1. 向协调者发送请求授权信息，1 次消息交互；
2. 协调者向程序发放授权信息，1 次消息交互；
3. 程序使用完临界资源后，向协调者发送释放授权，1 次消息交互。

因此，每个程序完成一次临界资源访问，需要进行 3 次消息交互。

不难看出，集中式算法的优点在于直观、简单、信息交互量少、易于实现，并且所有程序只需和协调者通信，程序之间无需通信。但是，这个算法的问题也出在了协调者身上。

一方面，协调者会成为系统的性能瓶颈。想象一下，如果有 100 个程序要访问临界资源，那么协调者要处理 100*3=300 条消息。也就是说，协调者处理的消息数量会随着需要访问临界资源的程序数量线性增加。

另一方面，容易引发单点故障问题。协调者故障，会导致所有的程序均无法访问临界资源，导致整个系统不可用。

因此，在使用集中式算法的时候，一定要选择性能好、可靠性高的服务器来运行协调者。

小结一下：集中式算法具有简单、易于实现的特点，但可用性、性能易受协调者影响。在可靠性和性能有一定保障的情况下，比如中央服务器计算能力强、性能高、故障率低，或者中央服务器进行了主备备份，主故障后备可以立马升为主，且数据可恢复的情况下，集中式算法可以适用于比较广泛的应用场景。

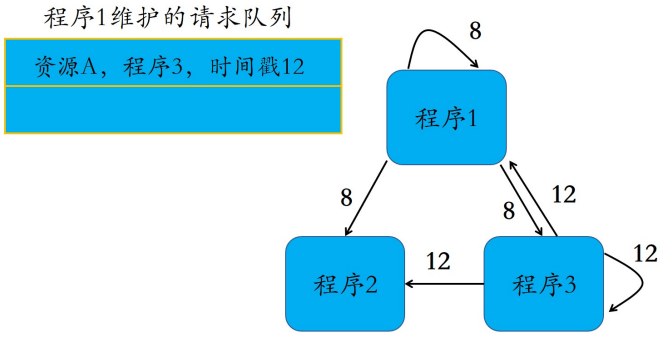
民主协商：分布式算法

既然引入协调者会带来一些问题，这时你可能就会想，不用协调者是否可以实现对临界资源的互斥访问呢？想象一下，当你需要使用自助咖啡机的时候，是不是可以先去征求其他人的意见，在确认其他人都没在使用也暂时不会使用咖啡机时，你就可以放心大胆地去泡制自己的咖啡了呢？

同理，我们可以把这套算法用于分布式系统。当一个程序要访问临界资源时，先向系统中的其他程序发送一条请求消息，在接收到所有程序返回的同意消息后，才可以访问临界资源。其中，请求消息需要包含所请求的资源、请求者的 ID，以及发起请求的时间。

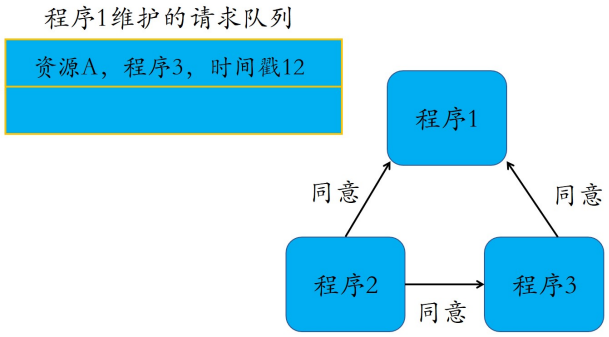
这，就是民主协商法。**在分布式领域中，我们称之为分布式算法，或者使用组播和逻辑时钟的算法。**

如图所示，程序 1、2、3 需要访问共享资源 A。在时间戳为 8 的时刻，程序 1 想要使用资源 A，于是向程序 2 和 3 发起使用资源 A 的申请，希望得到它们的同意。在时间戳为 12 的时刻，程序 3 想要使用资源 A，于是向程序 1 和 2 发起访问资源 A 的请求。



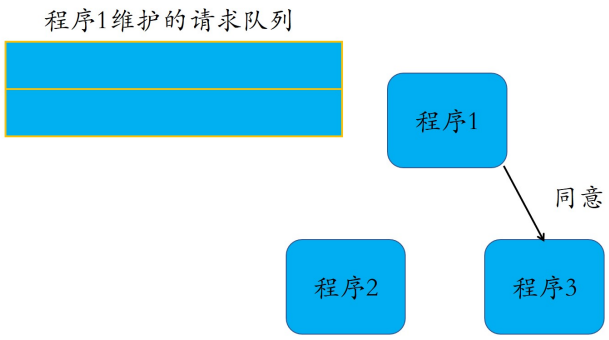
程序1和程序3差不多同一时间要访问共享资源A

如图所示，此时程序 2 暂时不访问资源 A，因此同意了程序 1 和 3 的资源访问请求。对于程序 3 来说，由于程序 1 提出请求的时间更早，因此同意程序 1 先使用资源，并等待程序 1 返回同意消息。



程序1的请求时间比程序3更早，获得所有授权，访问资源A

如图所示，程序 1 接收到其他所有程序的同意消息之后，开始使用资源 A。当程序 1 使用完资源 A 后，释放使用权限，向请求队列中需要使用资源 A 的程序 3 发送同意使用资源的消息，并将程序 3 从请求队列中删除。此时，程序 3 收到了其他所有程序的同意消息，获得了使用资源 A 的权限，开始使用临界资源 A 的旅程。



程序1释放资源A，程序3获得所有授权，访问资源A

从上述流程可以看出，一个程序完成一次临界资源的访问，需要进行如下的信息交互：

1. 向其他 n-1 个程序发送访问临界资源的请求，总共需要 n-1 次消息交互；
2. 需要接收到其他 n-1 个程序回复的同意消息，方可访问资源，总共需要 n-1 次消息交互。

可以看出，一个程序要成功访问临界资源，至少需要 2*(n-1) 次消息交互。假设，现在系统中的 n 个程序都要访问临界资源，则会同时产生 2n(n-1) 条消息。总结来说，**在大型系统中使用分布式算法，消息数量会随着需要访问临界资源的程序数量呈指数级增加，容易导致高昂的“沟通成本”。**

从上述分析不难看出，分布式算法根据“先到先得”以及“投票全票通过”的机制，让每个程序按时间顺序公平地访问资源，简单粗暴、易于实现。但，这个算法可用性很低，主要包括两个方面的原因：

> 当系统内需要访问临界资源的程序增多时，容易产生“信令风暴”，也就是程序收到的请求完全超过了自己的处理能力，而导致自己正常的业务无法开展。
>
> 一旦某一程序发生故障，无法发送同意消息，那么其他程序均处在等待回复的状态中，使得整个系统处于停滞状态，导致整个系统不可用。所以，相对于集中式算法的协调者故障，分布式算法的可用性更低。

针对可用性低的一种改进办法是，如果检测到一个程序故障，则直接忽略这个程序，无需再等待它的同意消息。这就好比在自助餐厅，一个人离开餐厅了，那你在使用咖啡机前，也无需征得他的同意。但这样的话，每个程序都需要对其他程序进行故障检测，这无疑带来了更大的复杂性。

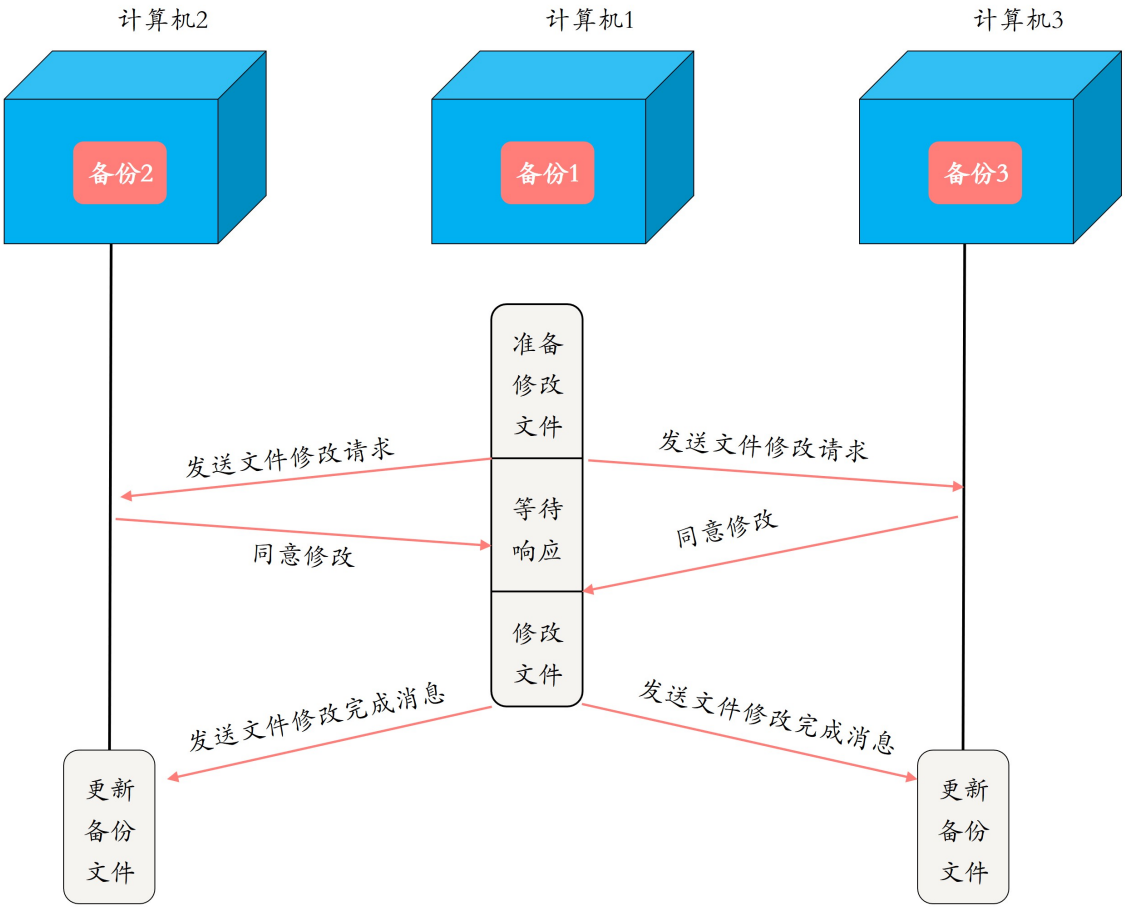
因此，**分布式算法适合节点数目少且变动不频繁的系统，且由于每个程序均需通信交互，因此适合 P2P 结构的系统。**比如，运行在局域网中的分布式文件系统，具有 P2P 结构的系统等。

那么，**在我们工作中，什么样的场景适合采用分布式算法呢?**

Hadoop 是我们非常熟悉的分布式系统，其中的分布式文件系统 HDFS 的文件修改就是一个典型的应用分布式算法的场景。

如下图所示，处于同一个局域网内的计算机 1、2、3 中都有同一份文件的备份信息，且它们可以相互通信。这个共享文件，就是临界资源。当计算机 1 想要修改共享的文件时，需要进行如下操作：

1. 计算机 1 向计算机 2、3 发送文件修改请求；
2. 计算机 2、3 发现自己不需要使用资源，因此同意计算机 1 的请求；
3. 计算机 1 收到其他所有计算机的同意消息后，开始修改该文件；
4. 计算机 1 修改完成后，向计算机 2、3 发送文件修改完成的消息，并发送修改后的文件数据；
5. 计算机 2 和 3 收到计算机 1 的新文件数据后，更新本地的备份文件。



HDFS 文件修改流程示意图

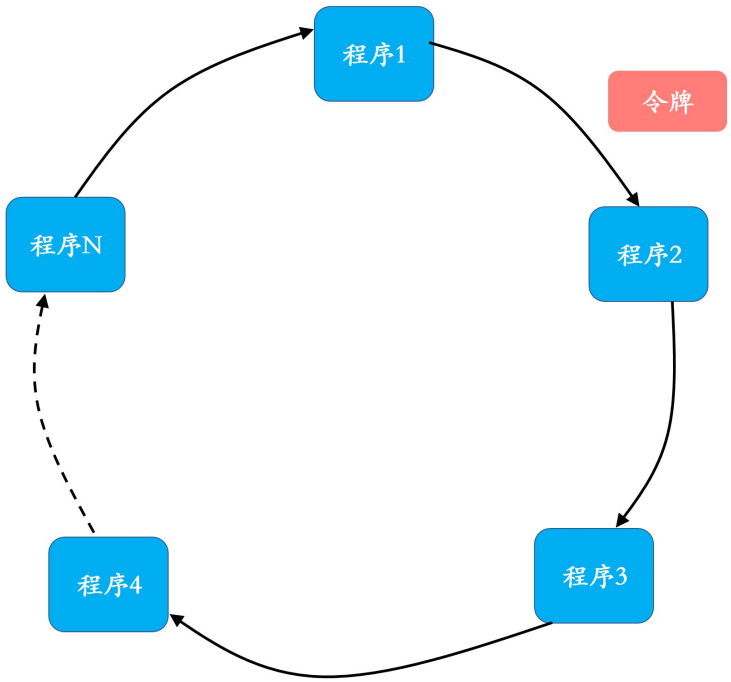
归纳一下：分布式算法是一个“先到先得”和“投票全票通过”的公平访问机制，但通信成本较高，可用性也比集中式算法低，适用于临界资源使用频度较低，且系统规模较小的场景。

轮值 CEO：令牌环算法

那么除了集中式算法、分布式算法以外，还有什么方法可以实现分布式互斥吗？答案是肯定的。毕竟，方法总比问题多。华为独创的轮值 CEO 其实就给了我们一个很好的启示。在华为的轮值 CEO 体系里，CEO 就是临界资源，同时只能有一个人担任，由多名高管轮流出任 CEO。

类似地，程序访问临界资源问题也可按照轮值 CEO 的思路实现。 如下图所示，所有程序构成一个环结构，令牌按照顺时针（或逆时针）方向在程序之间传递，收到令牌的程序有权访问临界资源，访问完成后将令牌传送到下一个程序；若该程序不需要访问临界资源，则直接把令牌传送给下一个程序。

在分布式领域，这个算法叫作令牌环算法，也可以叫作基于环的算法。为了便于理解与记忆，你完全可以把这个方法形象地理解为轮值 CEO 法。



令牌环算法示意图

因为在使用临界资源前，不需要像分布式算法那样挨个征求其他程序的意见了，所以相对而言，在令牌环算法里单个程序具有更高的通信效率。同时，在一个周期内，每个程序都能访问到临界资源，因此令牌环算法的公平性很好。

但是，不管环中的程序是否想要访问资源，都需要接收并传递令牌，所以也会带来一些无效通信。假设系统中有 100 个程序，那么程序 1 访问完资源后，即使其它 99 个程序不需要访问，也必须要等令牌在其他 99 个程序传递完后，才能重新访问资源，这就降低了系统的实时性。

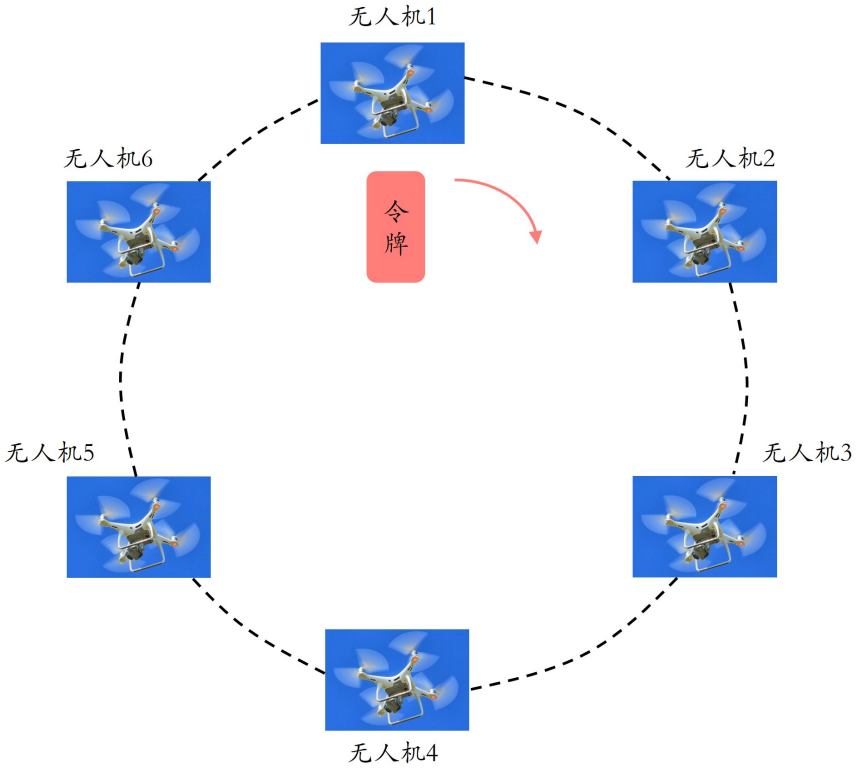
综上，**令牌环算法非常适合通信模式为令牌环方式的分布式系统**，例如移动自组织网络系统。一个典型的应用场景就是无人机通信。

无人机在通信时，工作原理类似于对讲机，同一时刻只能发送信息或接收信息。因此，通信中的上行链路（即向外发送信息的通信渠道）是临界资源。

如下图所示，所有的无人机组成一个环，按照顺时针方向通信。每个无人机只知道其前一个发送信息的无人机，和后一个将要接收信息的无人机。拥有令牌的无人机可以向外发送信

息，其他无人机只能接收数据。拥有令牌的无人机通信完成后，会将令牌传送给后一个无人机。

所有的无人机轮流通信并传输数据，从而消除了多个无人机对通信资源的争夺，使得每个无人机都能接收到其他无人机的信息，降低了通信碰撞导致的丢包率，保证了网络通信的稳定性，提高了多个无人机之间的协作效率。



无人机通信示意图

令牌环算法是一种更加公平的算法，通常会与通信令牌结合，从而取得很好的效果。特别是当系统支持广播或组播通信模式时，该算法更加高效、可行。

对于集中式和分布式算法都存在的单点故障问题，在令牌环中，若某一个程序（例如上图的无人机 2）出现故障，则直接将令牌传递给故障程序的下一个程序（例如，上图中无人机 1 直接将令牌传送给无人机 3），从而很好地解决单点故障问题，提高系统的健壮性，带来更好的可用性。但，这就要求每个程序都要记住环中的参与者信息，这样才能知道在跳过一个参与者后令牌应该传递给谁。

小结一下：令牌环算法的公平性高，在改进单点故障后，稳定性也很高，适用于系统规模较小，并且系统中每个程序使用临界资源的频率高且使用时间比较短的场景。

知识扩展：有适合大规模系统中的分布式互斥算法吗？

可以看到，上面提到的集中式、分布式和令牌环 3 个互斥算法，都不适用于规模过大、节点数量过多的系统。那么，什么样的互斥算法适用于大规模系统呢？

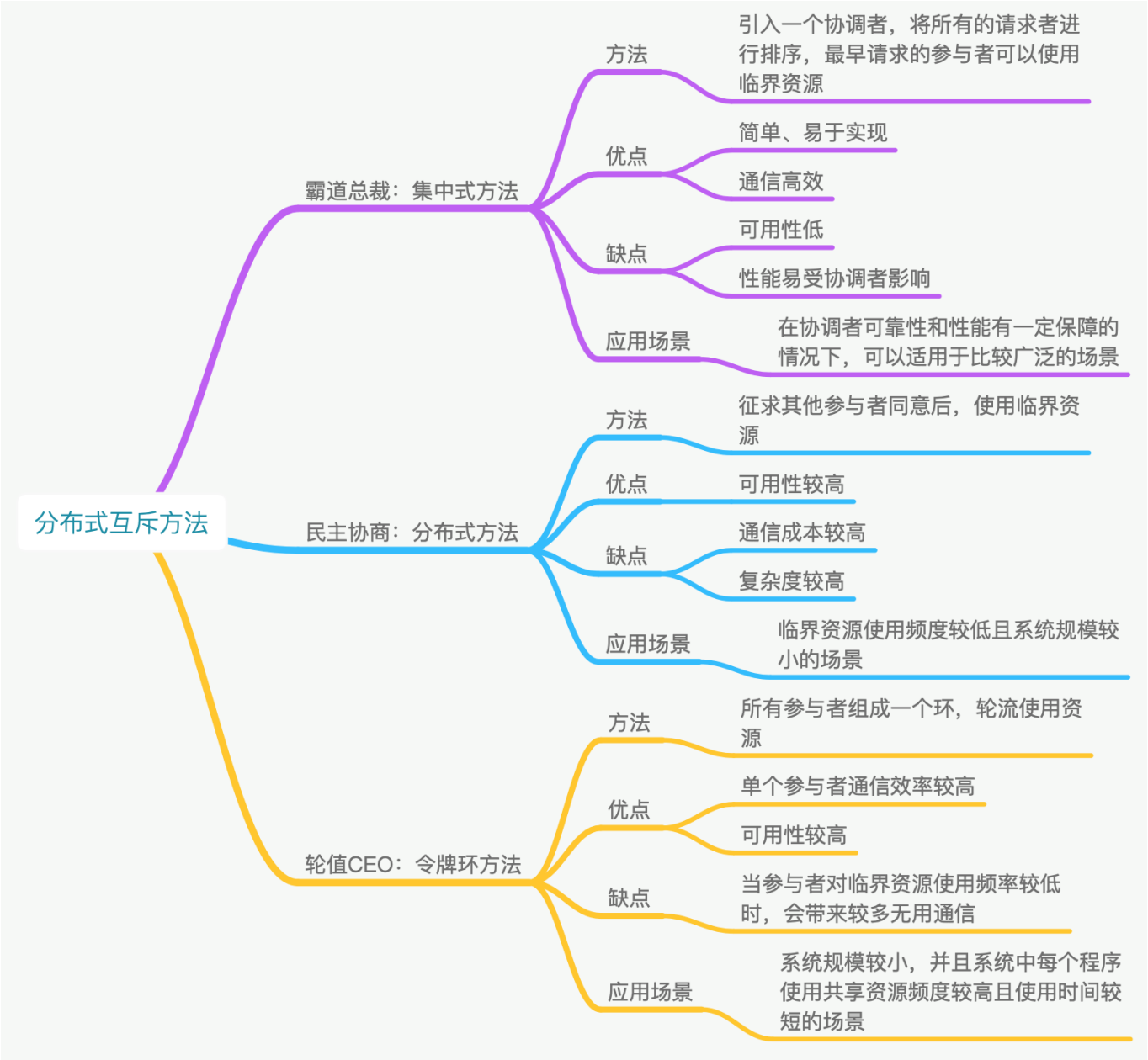
由于大规模系统的复杂性，我们很自然地想到要用一个相对复杂的互斥算法。时下有一个很流行的互斥算法，**两层结构的分布式令牌环算法，**把整个广域网系统中的节点组织成两层结构，可以用于节点数量较多的系统，或者是广域网系统。

我们知道，广域网由多个局域网组成，因此在该算法中，局域网是较低的层次，广域网是较高的层次。每个局域网中包含若干个局部进程和一个协调进程。局部进程在逻辑上组成一个环形结构，在每个环形结构上有一个局部令牌 T 在局部进程间传递。局域网与局域网之间通过各自的协调进程进行通信，这些协调进程同样组成一个环结构，这个环就是广域网中的全局环。在这个全局环上，有一个全局令牌在多个协调进程间传递。

总结

我首先结合生活中的案例，带你剖析了什么是分布式互斥，以及为什么需要分布式互斥。然后，我和你介绍了 3 类典型的分布式互斥方法，即：集中式算法、分布式算法，以及令牌环算法，并列举了对应的适用场景，相信你通过今天的学习，一定可以为你的场景选择一个合适的分布式互斥算法了，加油！

接下来，我把今天的内容通过下面的一张思维导图再全面总结下。



课后问题

最后，我想请你思考以下两个问题：

1. 你认为集中式算法、分布式算法和令牌环算法，还有什么可以改进的地方吗？
2. 传统单机上的互斥方法，为什么不能用于分布式环境呢？

我是聂鹏程，感谢你的收听，欢迎你在评论区给我留言分享你的观点，也欢迎你把这篇文章分享给更多的朋友一起阅读。我们下期再会！

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　02 | 分布式系统的指标：啥是分布式的三围

下一篇　04 | 分布式选举：国不可一日无君

精选留言 (19)

写留言

xingoo
2019-09-27

集中式没啥想法，分布式可以采用大多数投票而不是全部投票，令牌环可以为每个节点增加轮值权重，比如不常访问的两轮才轮一次。

传统的单机互斥靠加锁或者信号量，其实跟集中式差不多

展开

2　8

青莲
2019-09-28

集中式算法：可参照redis集群通信模式，通过hash key将大量的请求分散到不同的master,以处理大量请求,每个master由小集群主从节点来保障单点故障

分布式算法：分布式算法可在集群中过半数同意就识为其同意，降低通信数，如分布式选举场景
令牌环算法：可根据参与者使用频率列出权重，结合平滑加权轮询算法选出下一个参与...
展开

2

Geek_04d7ba
2019-09-27

为啥现在流行的分布式锁的实现比如zookeeper，redis都是用的集中式方法？是不是就是因为实现难度低？

1 1

leslie
2019-09-29

第一个问题是属于DB的弱点、、、、没办法回答：第二个应当是由于RMDB的强事务性ACID特性，而NO sql和MQ的只能顾到CAP特性中的2个，故而传统的互斥方法不能用于分布式系统。

北冥有鱼
2019-09-29

老师总结的很好，言简意赅，很精彩。
期待更新

zhaozp
2019-09-29

打卡文章学习：
1、分布式互斥对应于“四纵”的“分布式协同、调度”。互斥资源意味着不能同时访问，解决“让谁访问”的问题。
2、集中式算法：基于等待队列，协调者进行资源分配调度。但是程序本身会成为系统瓶颈，需要做主备。...
展开

汉超
2019-09-29

精通不但要有理论还要有丰富的实践，仅34节课怎么才能做到精通？

木叶
2019-09-29

算法思路比较清晰，整体上偏理论，能否结合典型的访问临界资源的场景举例，比如秒杀计数等场景

青铜5 周群力
2019-09-29

……则会同时产生 2n(n-1) 条消息。总结来说，在大型系统中使用分布式算法，消息数量会随着需要访问临界资源的程序数量呈指数级增加

——————————

这里应该是平方级增长吧 消息复杂度O(N^2)

展开

老男孩
2019-09-29

看了老师的讲解和大家的留言，真是受益匪浅。1）我还是觉得，集中式的分布式算法更好弄一些。单点问题和性能问题，是否可以使用协调者的集群方式？zookeeper？2）单机的互斥是通过管程或者说锁机制实现的，线程之间要想合理访问临界资源必须是同一把锁。分布式在不同机器上，用单机的方式无法拿到同一把锁，所有必须要有一个协调者的角色。

展开

Geek_54edc1
2019-09-28

2、分布式主要通过消息传递来进行协商，而单机主要通过锁或信号量，因此单机的互斥方法就不适用于分布式了

展开

Damon
2019-09-27

讲的挺好的,形象生动

展开

Zm
2019-09-27

还是觉得，集中式更好一些，可以把协调者改造为集群，增加可用性。

游弋云端
2019-09-27

协商机制中，可以采用一致性HASH解决资源路由的问题，避免像整个集群去广播查询资源的归属，同时通过资源归属的迁移可以支持节点扩缩容能力。

展开

gen_jin
2019-09-27

传统单机上的互斥方法，为什么不能用于分布式环境呢？回答：因为单机的临界资源在单机上，分布式的临界资源映射在不同的主机上，前者要解决多进程或多线程互斥访问单机上同一资源的问题，后者解决？

ashen
2019-09-27

集中式算法可以考虑把协调节点做成集群，参与者通过资源做一致性hash找对应的协调节点，这样可以一定程度解决单点问题。

展开

石维康
2019-09-27

传统单机上的互斥方法，为什么不能用于分布式环境呢？
因为在分布式场景下,很难保证操作的原子性

展开

zyh
2019-09-27

想请教下老师，hdfs作为hadoop生态的重要组件，经常用于大数据的处理。想问一下怎么理解说分布式算法适用于临界资源使用频度较低，且系统规模较小的场景。我个人的理解是较大规模的系统中也可以上hdfs。希望老师能够解答下，谢谢

展开

这个需求

做不了

啦啦啦

2019-09-27

不错不错

展开